Mobile Cloud Robotics as a Service with OCCIware

Philippe Merle, Christophe Gourdin, Nathalie Mitton

Mobile Cloud Robotics as a Service with OCCIware

Philippe Merle[1,2], Christophe Gourdin[1,2], Nathalie Mitton[1]
[1] Inria, France, firstname.lastname@inria.fr
[2] University Lille 1 CRIStAL UMR CNRS 9189, France

***Abstract*—We have recently witnessed the emerging of cloud computing on one hand and robotics platforms on the other hand. Naturally, these two visions have been merging to give birth to the Cloud Robotics paradigm in order to offer even more remote services. But such a vision is still in its infancy. Architectures and platforms are still to be defined to efficiently program robots so they can provide different services, in a standardized way masking their heterogeneity. This paper introduces OPEN MOBILE CLOUD ROBOTICS INTERFACE (OMCRI), a Robot-as-a-Service vision based platform, which offers a unified easy access to remote heterogeneous mobile robots. OMCRI encompasses an extension of the Open Cloud Computing Interface (OCCI) standard and a gateway hosting mobile robot resources. We then provide an implementation of OMCRI based on the open source model-driven Eclipse-based OCCIWARE tool chain and illustrates its use for three off-the-shelf mobile robots: Lego Mindstorm NXT, Turtlebot, and Parrot AR.Drone.**

***Index Terms*—Mobile Cloud Robotics, Robot-as-a-Service, Open Cloud Computing Interface, implementation.**

I. INTRODUCTION

With the advent of the Internet of Things (IoT) and robotics on one hand and of Cloud Computing on the other hand, people have witnessed a shift in the way they can interact and communicate with their things and their environment. Naturally, robotics platforms and cloud computing have been getting closer to lead to the concept of Cloud Robotics [13], [18], [19], [23], [30]. Such a new paradigm paves the way to various new applications and could allow for instance robot sharing for service providing, especially if we consider fleets of heterogeneous robots able to perform different tasks. We thus talk of the vision of Robot as a Service [6], [7]. Such a concept is gaining a lot of attention since it could be applied in many domains ranging from smart city management to rescue operations in remote areas [28] but it comes with a large sets of new challenges [13], [16], [18]. This paradigm is still in its infancy and new architectures are needed to manage robots. Current solutions are not yet user-oriented. Remote programming and control of heterogeneous robots are not always possible, which ends up as difficult tasks requiring advanced skills.

In this paper, we present OPEN MOBILE CLOUD ROBOTICS INTERFACE (OMCRI), an OCCI extension platform based on the Robot-as-a-Service paradigm, which allows the **easy remote programming** of heterogeneous robots, reducing access complexity for the user. OMCRI relies on the Open Cloud Computing Interface (OCCI), the open cloud standard [11], and proposes an abstraction of mobile robots and of the services they offer. OMCRI is modular and extensible. Specific commands to address each kind of robots is gathered in an independent module to plug into the OMCRI core. The adoption of a modular architecture eases the understanding, the development and the integration of the different robots programming and virtualization layer. OMCRI is innovative since to the best of our knowledge, it is the very first platform to provide simultaneously all the following features:

- OMCRI fills the gap between both cloud and robotics worlds by exposing a cloud-oriented programming interface,
- OMCRI is OCCI-standard compliant,
- OMCRI is scalable and extensible thanks to its modular architecture, and
- OMCRI is universal since it can support any kind of robots and deals with a set of heterogeneous robots.

This paper introduces the design of OMCRI, which consists of *(i)* an OCCI extension for modeling Mobile Cloud Robotics as a Service and *(ii)* the OMCRI gateway hosting mobile robot resources, and provides an implementation of OMCRI by using OCCIware [20], our open source model-driven Eclipse-based tool chain dedicated to OCCI. OMCRI supports any kind of robots. As an illustration of this, implementation is given for three different mobile robots: Lego Mindstorm NXT 2[1], Turtlebot 2[2], and Parrot AR.Drone[3]. Evaluation tests show that our module does not add latency, keeping server overhead low and remains stables over time and requests.

The remaining of the paper is as follows. In Section II, we motivate the need of such a cloud robotics platform through the description of application scenarios. Section III gives a background overview of OCCI. Section IV describes the design of OMCRI and especially its new OCCI extension for Mobile Cloud Robotics and the OMCRI gateway. Section V presents our implementation of OMCRI on top of the OCCIware tool chain and shows its relevance through the integration of three kinds of robots. Finally, we position our work with related approaches in Section VI before concluding and drawing some perspectives of future works in Section VII.

II. MOTIVATION

Imagine a disaster such as the Fukushima nuclear power plant that was damaged in 2011 by an earthquake and a tsunami. Human workers are better not to be sent to measure the damages because of radiations. Suppose that robots and drones are sent instead. Drones are essential to inspect the

[1] https://shop.lego.com/en-US/LEGO-MINDSTORMS-NXT-2-0-8547
[2] http://www.turtlebot.com
[3] https://www.parrot.com

most urgent places to reach from the air (if some victims are located for instance) communicate this information to the robots and then potentially guide them. Robots cooperate to explore the area on the ground. Several kinds of ground robots are needed based on the needs on the ground: They may need to climb, move, break, lift some stones, monitor radiation, rescue people, etc. Heterogeneity in both flying and ground robots is thus mandatory to perform well all tasks in such an application. Easy and standard way to remotely control them is also paramount.

But this is only an example of what could be achieved when heterogeneous robots cooperate [28]. But their potential is much wider since they can find applications in a lot of areas, such as civilian, military or industrial, at different scales. To name of few, we can refer to rescue operations, hostile environment exploration, infrastructure monitoring or destroying, fine-grained ground monitoring for agriculture, transportation enhancement, etc. In all these applications, having fleets of heterogeneous robots is a great asset but this potentially goes with an increased complexity in controlling them. Therefore, this is paramount to mask these difficulties and offer to the user a unified view of all robots, together with simplified tools and a unique interface to remotely address and control them. This is the purpose of OMCRI.

III. Background on OCCI

OMCRI is an extension of the Open Cloud Computing Interface (OCCI), the open cloud standard [11] specified by the Open Grid Forum (OGF). OCCI defines a RESTful[4] Protocol and API[5] for all kinds of management tasks on any kind of cloud resources, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and especially Mobile Robotics as a Service as proposed in this paper. In order to be modular and extensible, OCCI is delivered as a set of specification documents[6] divided into the four following categories as illustrated in Fig. 1:

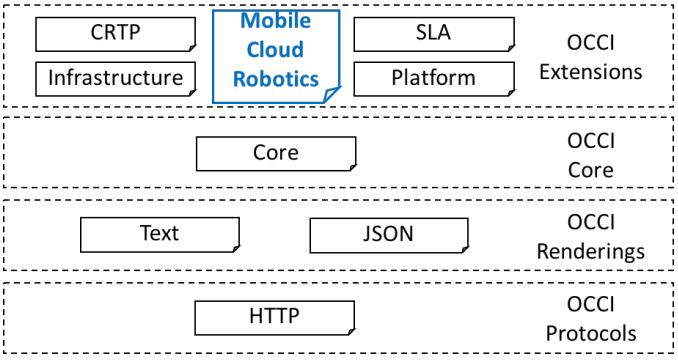

Fig. 1. OCCI Specifications (the dark blue box is our OMCRI contribution).

- **OCCI Core**: The OCCI Core specification [26] defines a RESTful-oriented model. The main concepts of this model are `Resource`, `Link`, `Kind`, `Mixin`, `Attribute`, and `Action`. `Resource` is the root abstraction of any cloud resource, such as a virtual machine, a network, an application, and a mobile robot. `Link` represents a relation between two resources, such as a virtual machine connected to a network and an application hosted by a virtual machine. Each OCCI entity (resource and link) owns zero or more `Attributes`, such as its unique identifier, the host name of a virtual machine, the Internet Protocol address of a network. Each OCCI entity has zero or more `Actions` representing business specific behaviors, such as start/stop a virtual machine, and up /down a network. Each OCCI entity is strongly typed by `Kind` and `Mixins`. `Kind` represents the immutable type of OCCI entities and defines allowed attributes and actions. Single inheritance between `Kinds` allows us to factorize attributes and actions common to several kinds. `Mixin` represents cross-cutting attributes and actions that can be dynamically added to an OCCI entity. `Mixin` can be applied to zero or more `Kinds` and can depend from zero or more other `Mixins`. For more details on the OCCI Core Model, readers can refer to [20]. OCCI Core can be interacted with over protocols/renderings and is expandable through extensions.
- **OCCI Protocols**: Each OCCI Protocol specification describes how a particular network protocol can be used to interact with the OCCI Core Model. Multiple protocols can interact with the same instance of the OCCI Core Model. Currently, only the OCCI HTTP Protocol [25] has been defined but other OCCI protocols could be proposed in the future such as AMQP[7].
- **OCCI Renderings**: Each OCCI Rendering specification describes a particular rendering of the OCCI Core Model. Multiple renderings can interact with the same instance of the OCCI Core Model and will automatically support any OCCI extension. Currently, both OCCI Text [10] and JSON[8] [27] renderings have been defined.
- **OCCI Extensions**: Each OCCI Extension specification describes a particular extension of the OCCI Core Model for a specific application domain, and thus defines a set of domain-specific `Kinds` and `Mixins`. OCCI Infrastructure [21] is dedicated to IaaS and defines `Compute`, `Network`, `NetworkInterface`, `Storage`, `StorageLink` kinds, `IPNetwork` and `Credentials` mixins. OCCI Compute Resource Templates Profile (CRTP) [8] defines a set of well-defined instances of the `Compute` resource kind, such as `small`, `medium`, and `large` mixins. OCCI Platform [22] is dedicated to PaaS and defines `Application`, `Component`, and `ComponentLink` kinds. OCCI Service Level Agreements [14] is dedicated to define SLA in OCCI. **This paper proposes a new OCCI extension for mobile cloud robotics**.

[4]REpresentation State Transfer [12]
[5]Application Programming Interface
[6]Available at http://occi-wg.org/about/specification/.
[7]Advanced Message Queuing Protocol
[8]JavaScript Object Notation

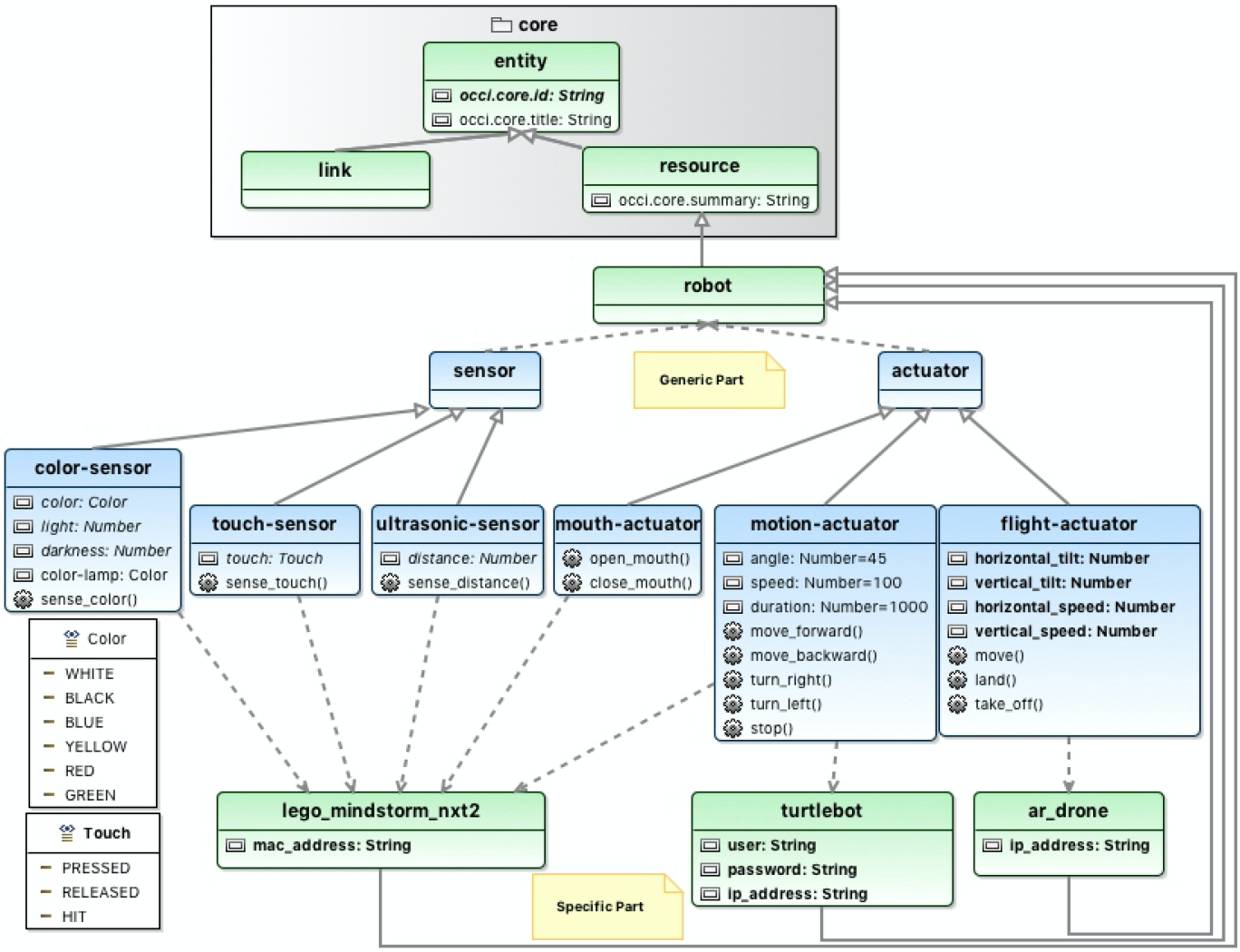

Fig. 2. OMCRI Extension Diagram.

IV. Open Mobile Cloud Robotics Interface

Open Mobile Cloud Robotics Interface features two main components *(i)* an OCCI extension for modeling Mobile Cloud Robotics as a Service and *(ii)* the OMCRI gateway hosting mobile robot resources. We detail these both components in the two next sections.

A. OCCI Extension for Mobile Cloud Robotics

Our OCCI extension for Mobile Cloud Robotics (called the OMCRI extension) is presented in the form of a diagram in Fig. 2. This diagram was designed with OCCIware Studio [29], our open source model-driven based integrated development environment dedicated to OCCI. The OCCI Core Model is represented by the grey box. Kinds and mixins are represented by green and blue boxes, respectively. Type inheritance is represented by full arrows. The application of a mixin to a kind is represented by dotted arrows. Mandatory attributes are in bold and immutable attributes are in italics. Enumeration data types are represented by white boxes.

Our OMCRI extension is split into two parts: 1) a generic part at the top of Fig. 2 and 2) a specific part at the bottom of Fig. 2. The generic part defines abstract robot agnostic types (kinds and mixins). The specific part defines concrete robot-specific types. OMCRI is extensible by design to support any kind of ground and flying robots, whatever their communication and core technology. However, in this paper, the specific part is limited to three particular mobile robots: Lego Mindstorm NXT 2, Turtlebot 2 and Parrot AR.Drone.

The generic part of the OMCRI extension is composed of one kind and two mixins:

- **Robot**: This kind is the base type of any robot in OMCRI, *i.e.*, specific robot kinds inherit from `robot`. As `robot` inherits from the `resource` kind, any robot owns transitively three attributes: *(i)* `occi.core.id` the immutable mandatory unique identifier of the robot, *(ii)* `occi.core.title` a mutable optional string entitled the robot and *(iii)* `occi.core.summary` a mutable optional text describing the robot.

- **Sensor**: This mixin is the base type of any robot sensor in OMCRI, *i.e.*, specific sensor types inherit from `sensor`. A robot can have zero or more sensors.
- **Actuator**: This mixin is the base type of any robot actuator in OMCRI, *i.e.*, specific actuator types inherit from `actuator`. A robot can have zero or more actuators.

The robot-specific part of the OMCRI extension is composed of three kinds (one for each targeted robot) and six mixins (three robot-specific sensors and three actuators):

- **Lego Mindstorm NXT 2**: This kind represents Lego Mindstorm NXT 2 robots and defines one specific mandatory attribute: `mac_address` is the MAC address of a Lego Mindstorm NXT 2 robot. This address must be configured at the creation time of the robot resource and is used later by OMCRI to interact via Bluetooth with the robot.
- **Turtlebot**: This kind represents Turtlebot 2 robots and defines three specific mandatory attributes. Both attributes `user` and `password` are the security credentials to connect to the robot. Attribute `ip_address` is the IP address to interact with the robot. These attributes must be configured at the creation time of the robot resource and are used later by OMCRI to securely connect/interact with a Turtlebot 2 robot.
- **AR.Drone**: This kind represents Parrot AR.Drone robots and defines one specific mandatory attribute. Attribute `ip_address` is the IP address of a drone. This address must be configured at the creation time of the drone and is used later by OMCRI to interact via WiFi with the drone.
- **Color Sensor**: This mixin represents the color sensor of a Lego Mindstorm NXT 2 robot, *i.e.*, `color-sensor` is applied to `lego-mindstorm-nxt2`. This sensor provides three immutable sensing attributes: `color` is the last sensed color, `light` is the last sensed light, and `darkness` is the last sensed darkness. This sensor can sense six distinct colors (white, black, blue, yellow, red, and green) as defined by the `Color` enumeration data type. Moreover, this sensor can act as a lamp and the color of this lamp is configured by the mutable `color-lamp` attribute. The `sense_color` action does the sensing and updates the three sensing attributes accordingly.
- **Touch Sensor**: This mixin represents the touch sensor of a Lego Mindstorm NXT 2 robot, *i.e.*, `touch-sensor` is applied to `lego-mindstorm-nxt2`. This sensor provides one immutable sensing attribute: `touch` is the last sensed touch state. The three possible touch status (pressed, released, and hit) are defined by the `Touch` enumeration data type. The `sense_touch` action does the sensing and updates the `touch` attribute accordingly.
- **Ultrasonic Sensor**: This mixin represents the ultrasonic sensor of a Lego Mindstorm NXT 2 robot, *i.e.*, `ultrasonic-sensor` is applied to `lego-mindstorm-nxt2`. This sensor measures the distance between the robot and obstacles. The last sensed distance is stored into the `distance` attribute. The `sense_distance` action does the sensing and updates the `distance` attribute accordingly.
- **Mouth Actuator**: This mixin represents the mouth actuator of a Lego Mindstorm NXT 2 robot, *i.e.*, `mouth-actuator` is applied to `lego-mindstorm-nxt2`. This actuator has no configurable attributes and provides two actions: `open_mouth` and `close_mouth` to open and close the mouth of the robot, respectively.
- **Motion Actuator**: This mixin represents the motion actuator of any robot able to move. Let's note that this mixin is applied to two kinds of robots: `lego-mindstorm-nxt2` and `turtlebot`. This mixin defines three configurable attributes: `angle` is the angle when the robot turns (45 degree by default), `speed` defines the motion speed when the robot moves (100 by default), and `duration` is the motion duration when the robot moves (1 second by default). Fives actions are allowed on mobile robots: `move_forward`, `move_backward`, `turn_right`, `turn_left`, and `stop`.
- **Flight Actuator**: This mixin represents the flight actuator of any robot able to fly. In this paper, this mixin is only applied to `ar_drone` robots. Four mandatory configurable attributes are defined: `horizontal_tilt` is the horizontal flight tilt of the drone, `vertical_tilt` is the vertical flight tilt of the drone, `horizontal_speed` is the horizontal flight speed of the drone, and `vertical_speed` is the vertical flight speed of the drone. Three actions can be executed on drones: `move`, `land`, and `take_off`.

B. OMCRI Gateway Architecture

At runtime, end users (or programs) interact with mobile robots through the networked OMCRI gateway, as illustrated in Fig. 3. The OMCRI gateway is composed of three parts:

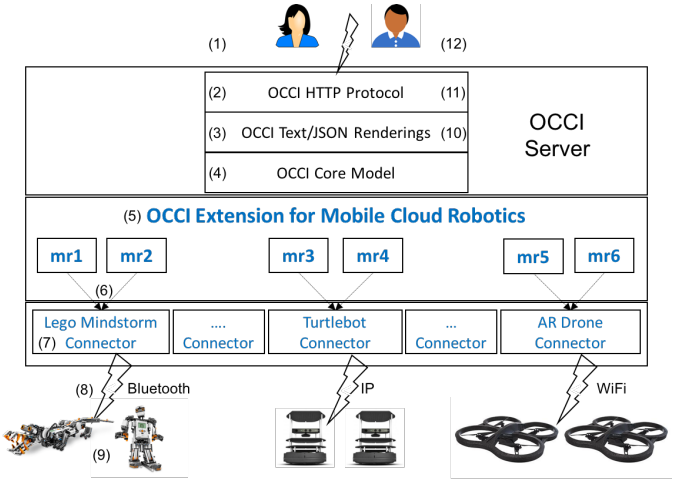

Fig. 3. OMCRI Gateway Architecture.

- **OCCI Server**: The OMCRI gateway embeds an OCCI server implementing the following OCCI specifications:
  - OCCI HTTP Protocol [25],
  - OCCI Text/JSON Renderings [10], [27], and
  - OCCI Core Model [26].
- **OMCRI Extension**: The OMCRI gateway supports the OCCI extension for Mobile Cloud Robotics presented in Section IV-A.
- **OMCRI Connectors**: The OMCRI gateway contains a set of connectors. Each connector is dedicated to a specific robot, *e.g.*, Lego Mindstorm NXT 2, Turtlebot, and AR Drone. The OMCRI gateway is extensible by design: Supporting a new kind of robots consists of adding a new connector.

Users send their HTTP requests to the OMCRI gateway and wait for a reply (Step 1 in Fig. 3). In the OMCRI gateway, the processing of a user's request consists of managing the OCCI HTTP protocol (Step 2), decoding the HTTP request body according to its text or JSON format (Step 3), forwarding the request to the OCCI Core Model (Step 4), controlling if the request is allowed by the OMCRI extension (Step 5), calling the connector related to the targeted robot (Step 6), preparing the request to send to the robot (Step 7), communicating with the robot via its associated network protocol (Step 8), processing the request by the robot (Step 9), encoding the HTTP reply body (Step 10), and return the reply to the user (Step 11). Then the user processes the reply (Step 12).

V. Implementing OMCRI with OCCIware

A. Overview

The implementation of OMCRI is built on top of our open source model-driven Eclipse-based OCCIware tool chain [29] composed of two main parts: *(i)* OCCIware Studio[9] an integrated development environment for OCCI and *(ii)* OCCIware Runtime[10] a full OCCI-compliant server implementing OCCI Core [26], OCCI HTTP Protocol [25], and OCCI Text/JSON Renderings [10], [27] specifications.

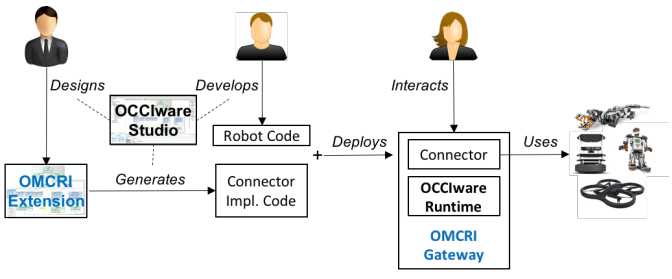

Fig. 4. Implementing OMCRI with OCCIware.

As illustrated in Fig. 4, a designer defines OMCRI types for its mobile robots (*i.e.*, specific `robot` kinds, their `sensor` and `actuator` mixins, and required data types) with OCCIware Studio as discussed in Section IV-A. Then, OCCIware Studio generates the implementation of a connector for each modeled mobile robot. A developer completes each generated connector with robot-specific code implementing the business logic to interact with the mobile robot as discussed in Section V-B. The connector is then deployed on the OMCRI gateway, which embeds OCCIware Runtime to process users' OCCI HTTP requests as discussed in Section IV-B. Users interact with the mobile robots as presented in Section V-C.

B. Implementing OMCRI Connectors

To implement the OMCRI connector associated to a mobile robot, the developer must complete the code generated by OCCIware Studio. Listing 1 presents the generated connector class for Lego Mindstorm NXT 2.

Listing 1. Generated connector for Lego Mindstorm NXT 2.

```
public class LegoMindstormNxt2Connector
    extends LegoMindstormNxt2Impl
{
    public void occiCreate() {/* TBC */}
    public void occiRetrieve() {/* TBC */}
    public void occiUpdate() {/* TBC */}
    public void occiDelete() {/* TBC */}

    public void sense_color() {/* TBC */}
    public void sense_touch() {/* TBC */}
    public void sense_distance() {/* TBC */}
    public void open_mouth() {/* TBC */}
    public void close_mouth() {/* TBC */}
    public void move_forward() {/* TBC */}
    public void move_backward() {/* TBC */}
    public void turn_right() {/* TBC */}
    public void turn_left() {/* TBC */}
    public void stop() {/* TBC */}
}
```

The generated `LegoMindstormNxt2Connector` Java class extends another generated class `LegoMindstormNxt2Impl`, which contains all the OCCIware Runtime specific code. The connector class contains four OCCI specific callback methods and all the robot specific actions that must be completed (TBC). The callback methods are called by OCCIware Runtime to signal that the OCCI resource *(i)* was created (`occiCreate`), *(ii)* will be retrieved (`occiRetrieve`), *(iii)* was updated (`occiUpdate`), and *(iv)* will be deleted (`occiDelete`). These callbacks can open a network connection to the mobile robot, consult the robot's state, update its state, and close the connection, respectively. The developer must implement all the actions defined in both the OCCI kind of the robot (`lego_mindstorm_nxt2`) and all its applied mixins (`color-sensor`, `touch-sensor`, `ultrasonic-sensor`, `mouth-actuator`, and `motion-actuator`). The implementation of each action consists of forwarding the called action to the mobile robot through the appropriate networked robot protocol (*e.g.*, Bluetooth for Lego Mindstorm NXT 2, IP for Turtlebot, WiFi for AR.Drone). The

[9]Available at https://github.com/occiware/OCCI-Studio
[10]Available at https://github.com/occiware/MartServer

implementation details are not discussed in this paper due to space limit[11].

C. Interacting with mobile robots

Users interact with mobile robots by sending OCCI HTTP requests to the OMCRI gateway (see Section IV-B). Then, we show three typical users' requests: creation of a mobile robot resource, updating its attributes, and executing an action of the robot.

Listing 2. OMCRI request for creating a Lego Mindstorm NXT 2 resource.

```
{
    "title": "robot vehicule claptap",
    "summary": "robot claptap car",
    "kind":
        "http://omcri#lego_mindstorms_nxt2",
    "mixins": [
       "http://omcri#motion-actuator",
       "http://omcri#color-sensor"
    ],
    "attributes": {
       "mac_address": "00:16:53:10:10:C3",
       "speed": 1,
       "color-lamp": "RED"
    },
    "location": "lego/myRobot"
}
```

To create a mobile robot resource, users send an HTTP PUT request with a JSON body as illustrated in Listing 2. The body of a creation request must contain the robot `kind` (`lego_mindstorms_nxt2`), its associated `mixins` (*e.g.*, `motion-actuator` and `color-sensor`), and the initial value of mandatory attributes (`mac_address` defined in bold in Fig. 2). The request could also contain the value of other optional attributes (both `occi.core.title` and `occi.core.summary` of `resource` kind, `speed` of `motion-actuator` mixin, and `color-lamp` of `color-sensor` mixin in Fig. 2). The last `location` field defines the HTTP path (`lego/myRobot`) to interact with the mobile robot later.

Listing 3. OMCRI request for updating a Lego Mindstorm NXT 2 resource.

```
{
  "location": "lego/myRobot",
  "attributes": {
    "duration": 2000
  }
}
```

To update a mobile robot resource, users send an HTTP POST request with a JSON body as illustrated in Listing 3. The body of an updating request must contain the location of the robot to update (`lego/myRobot`) and the list of attributes to update (*e.g.*, `duration` to update to 2000 milliseconds).

Listing 4. Executing `move_forward` on a Lego Mindstorm NXT 2 resource.

```
{
  "location": "lego/myRobot",
  "action": "http://omcri/motion-actuator/
      action#move_forward"
}
```

To execute an action on a mobile robot resource, users send an HTTP POST request with a JSON body as illustrated in Listing 4. The body of an execution request must contain the location to the robot (`lego/myRobot`), the action to execute (`move_forward` of mixin `motion-actuator`), and action parameters (if required by the action). Thus Listing 4 requests the previously created Lego Mindstorm NXT 2 robot to move forward during two seconds (the last value of the `duration` attribute).

D. OMCRI Gateway Performance Evaluation

The objective of this evaluation is to measure the performance, stability, and scalability of the OMCRI gateway, and determinate the overhead introduced by the OMCRI gateway when executing an action on a mobile robot.

The OMCRI gateway has been installed on a MacBook Pro with Intel Core I7 2.2 Ghz 8 cores, 16 GB 1600 Mhz DDR3 of RAM, 250 GB SSD of disk, running MacOS Sierra operating system and Oracle Java Runtime Environment version 8. The measures were taken with Apache Jmeter[12] version 3.1, an open source software to evaluate HTTP performance and to load HTTP servers. To avoid network latency, the OMCRI gateway and JMeter have been launched on the same machine. The OMCRI gateway hosts the three connectors for Lego Mindstorm NXT 2, Turtlebot, and AR Drone. In this paper, we only present and analyse the measures for the Lego Mindstorm NXT 2 connector. However the measures for other connectors are similar.

We run *ten shots*. Each slot includes 1,000 create, 10,000 update, 10,000 retrieve and 30 execute action requests. Based on these 210,300 requests, we compute average response time per request, median time, standard deviation, variance, and the number of requests per second. Fig. 5 summarizes all these measures and Table I shows them as a histogram.

Our analysis of these observed measures is:

1) As no request failed during all the ten shots, we can consider that our implementation of the OMCRI gateway is **robust** and **reliable**.

2) As the average response time is very close to median time and both standard deviation and variance of create, update, and retrieve requests are extremely low, we can consider that the performance of the OMCRI gateway is **extremely stable** and **predictable**.

3) The OMCRI gateway **scales well** until 1,000 resources, which already form an extremely large fleet of mobile robots.

4) The OMCRI gateway provides a **high throughput** for update and retrieve requests (*i.e.*, 2,049 and 2,386 requests per second, respectively). The throughput of create requests (249) is lower. This is due to the fact that creating a new resource requires memory allocation and verification of the

[11]A prototype of the three connectors is available at https://github.com/PFECloudRobotics2017.

[12]http://jmeter.apache.org/

TABLE I
OMCRI GATEWAY PERFORMANCE EVALUATION

OMCRI Request	Requests/shot	Average Response Time (ms)	Median (ms)	Standard Deviation	Variance	Requests/sec.
Create a Lego Mindstorm NXT 2 resource	1,000	4.015	3.995	0.119	0.014	249
Update a Lego Mindstorm NXT 2 resource	10,000	0.488	0.490	0.004	0.0000178	2,049
Retrieve a Lego Mindstorm NXT 2 resource	10,000	0.419	0.420	0.006	0.000032	2,386
Execute action `turn_left`	30	65.526	65.135	1.873	3.51	15
OMCRI overhead for executing action `turn_left`		1.134	1.130	0.126	0.016	882

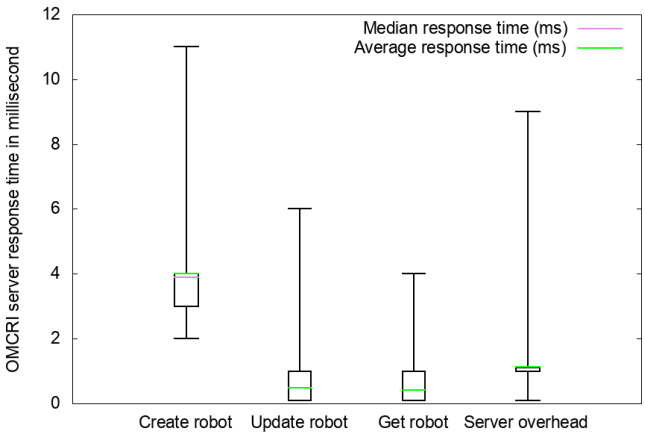

Fig. 5. OMCRI Gateway Response Time.

existence and compatibility[13] of the requested `kind` and applied `mixins`, and checking that users' requests contain all mandatory attributes.

5) The **overhead** introduced by the OMCRI gateway **is very low** (near 1.1 ms) compared to the total time needed by the robot to execute the requested action (near to 65 ms to turn left a Lego Mindstorm NXT 2). Of course, the overhead depends on the requested action and the kind of robots. However, for action `turn_left`, the overhead is near 1.73%.

VI. RELATED WORK

As already mentioned, the concept of Cloud Robotics have slightly emerged recently [18]. The community agreed that getting robots and cloud closer will allow for a new world of opportunities. First attempts were mainly proposing data sharing between robots, creating a so-called social network for robots, allowing one to develop their own services at home on robots [2].

Exploiting the full potential of the cloud to not only share data but also offload complex computing has emerged with [9], [24]. Then, the vision of Robot as a Service (RaaS) has emerged with its sets of challenges [13], [16], [19]. Several platforms have started to emerge, providing more or less open robot programming tools. The authors of [30] provide a platform that allows the remote control of robots through shape recognizing software to overcome the absence of camera on robots. [7] offers a web-based robot programming interface, which is actually a great asset since available from almost everywhere and user-friendly. Both of these latter approaches consider the concept of Robot as a Service but they only consider fleets of homogeneous robots. [6] enables the control of different kinds of robots such as Lego Mindstorms and iRobots among others but it relies on proprietary tools for their programming. Complete platforms have been proposed like DaVinci [1], [5], [15], [17] and Rapyuta [23].

DaVinci (Distributed Agents with Collective Intelligence) was one of the first implementation of a collaborative platform. It showed the advantages of cloud computing by paralyzing a SLAM algorithm using a Hadoop cluster. Although innovative and promising, it was not very reliable neither scalable since it was not supporting too big data volumes and not publicly available. [1] offers to execute robot behaviors such as vision and speech algorithms on compatible robots in the cloud. [15] uses Google's Object Recognition Engine to recognize and grasp common household objects. These two approaches allow for the use of heterogeneous robots but unfortunately are not publicly available.

Rapyuta [23] is certainly the most similar approach than OMCRI. Rapyuta is the RoboEarth Cloud Engine, an open source cloud robotics framework. The RoboEarth project [3] offers a Cloud Robotics infrastructure, which aims to include everything needed to close the loop from the robot to the cloud and back to the robot. Each robot connected to Rapyuta has a secured computing environment giving it the ability to move its heavy computation into the cloud. Similarly to OMCRI, it could be applied in a large set of scenario, including rescue operations [28]. RoboEarth provides an open-source Cloud Robotics framework that allows robots to share knowledge via a WWW-style database and access powerful robotics cloud services. It thus provides an open-source implementation for a robotics platform able to manage different kinds of robots. However, Rapyuta does not rely on cloud standards but rather integrates ad hoc implementation of modules.

VII. CONCLUSION

This paper presented OMCRI encompassing an OCCI extension for modeling Mobile Cloud Robotics as a Service and the OMCRI gateway for hosting mobile robot resources. OMCRI is modular and extensible. It can support any kind of robots. We used the OCCIware tool chain to provide an implementation for three off-the-shelf mobile robots to illustrate its ease of use and how robot heterogeneity is made

[13] *e.g.*, checking that the `flight-actuator` mixin is not applied to a `lego_mindstormnxt2` resource.

transparent for the user. Evaluation tests have shown that the gateway overhead remains low and that OMCRI does not add any latency. As perspectives, we forecast to extend OMCRI for supporting other kinds of robots and make them cooperate at a large scale. Implementations and then different scenarios will be able to be experimented over the FIT IoT-LAB platform[14] [4], emulating different application scenarios as described in Section II. Another perspective is to size the OMCRI gateway in order to directly embed it into mobile Internet of Things such as autonomous vehicles either for intelligent transportation systems or smart agriculture monitoring (vehicles can be farming vehicles), personal care or event shooting [31]. Finally, we will propose OMCRI to Open Grid Forum in order to standardize an OCCI extension for mobile robots.

ACKNOWLEDGMENTS

We thank Lucas Delvallet, Quentin Delvallet, and Florian Doublet for the implementation of the first prototype of OMCRI. This work is partially supported by both the CPER/FEDER DATA grant and the OCCIware research and development project (www.occiware.org) funded by French Programme d'Investissements d'Avenir (PIA).

REFERENCES

[1] Gostainet. http://www.gostainet.com.

[2] My Robots. https://en.wikipedia.org/wiki/MyRobots.

[3] RoboEarth. http://roboearth.ethz.ch.

[4] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, et al. FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464. IEEE, 2015.

[5] Rajesh Arumugam, Vikas Reddy Enti, Liu Bingbing, Wu Xiaojun, Krishnamoorthy Baskaran, Foong Foo Kong, A.Senthil Kumar, Kang Dee Meng, and Goh Wai Kit. DaVinci: A Cloud Computing Framework for Service Robots. In *Proceedings of 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3084–3089, Anchorage, USA, 2010. IEEE.

[6] Yinong Chen and Hualiang Hu. Internet of Intelligent Things and Robot as a Service. *Simulation Modelling Practice and Theory*, 34:159–171, 2013.

[7] Yinong Chen and Zhizheng Zhou. Robot as a Service in Computing Curriculum. In *Proceedings of 2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 156–161, Taichung, Taiwan, 2015. IEEE.

[8] Michel Drescher, Boris Parák, and David Wallom. OCCI Compute Resource Templates Profile. Recommandation GFD-R-P.222, Open Grid Forum, October 2016.

[9] Zhihui Du, Weiqiang Yang, Yinong Chen, Xin Sun, Xiaoying Wang, and Chen Xu. Design of a Robot Cloud Center. In *Proceedings of IEEE Twelfth International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 269–275, Kobe, Japan, 2011. IEEE.

[10] Andy Edmonds and Thijs Metsch. Open Cloud Computing Interface – Text Rendering. Recommandation GFD-R-P.229, Open Grid Forum, October 2016.

[11] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. Toward an Open Cloud Standard. *IEEE Internet Computing*, 16(4):15–25, 2012.

[12] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[13] D Gowrimol. A Study on Cloud Robotics and Automation. *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, 5(4):25–26, 2016.

[14] Gregory Katsaros. Open Cloud Computing Interface – Service Level Agreements. Recommandation GFD-R-P.228, Open Grid Forum, October 2016.

[15] Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. Cloud-Based Robot Grasping with the Google Object Recognition Engine. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 4263–4270, Karlsruhe, Germany, 2013. IEEE.

[16] Ben Kehoe, Sachin Pati, Pieter Abbeel, and Ken Goldberg. A Survey of Research on Cloud Robotics and Automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015.

[17] Anis Koubâa. A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Cloud. In *Proceedings of International Conference on Architecture of Computing Systems (ARCS)*, pages 196–208, Lübeck, Germany, 2014. Springer.

[18] Daniel Lorencik and Peter Sincak. Cloud Robotics: Current Trends and Possible Use as a Service. In *Proceedings of 11th IEEE International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 85–88, Herlány, Slovakia, 2013. IEEE.

[19] Xinqiang Ma and Yi Huang. Research on Mobile Cloud Robotics based on Cloud Computing. In *Proceedings of International Forum on Management, Education and Information Technology Application (IFMEITA)*, Guangzhou, China, 2016.

[20] Philippe Merle, Olivier Barais, Jean Parpaillon, Noël Plouzeau, and Samir Tata. A Precise Metamodel for Open Cloud Computing Interface. In *Proceedings of 8th IEEE International Conference on Cloud Computing, CLOUD 2015*, pages 852–859. IEEE, June 2015.

[21] Thijs Metsch, Andy Edmonds, and Boris Parák. Open Cloud Computing Interface – Infrastructure. Recommandation GFD-R-P.224, Open Grid Forum, October 2016.

[22] Thijs Metsch and Mohamed Mohamed. Open Cloud Computing Interface – Platform. Recommandation GFD-R-P.227, Open Grid Forum, October 2016.

[23] Gajamohan Mohanarajah, Dominique Hunziker, Raffaello D´Andrea, and Markus Waibel. Rapyuta: A Cloud Robotics Platform. *IEEE Transactions on Automation Science and Engineering*, 12(2):481–493, 2015.

[24] Carla Mouradian, Fatima Zahra Errounda, Fatna Belqasmi, and Roch Glitho. An Infrastructure for Robotic Applications as Cloud Computing Services. In *Proceedings of 2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 377–382, Seoul, Korea, 2014. IEEE.

[25] Ralf Nyrén, Andy Edmonds, Thijs Metsch, and Boris Parák. Open Cloud Computing Interface – HTTP Protocol. Recommandation GFD-R-P.223, Open Grid Forum, October 2016.

[26] Ralf Nyrén, Andy Edmonds, Alexander Papaspyrou, Thijs Metsch, and Boris Parák. Open Cloud Computing Interface – Core. Recommendation GFD-R-P.221, Open Grid Forum, October 2016.

[27] Ralf Nyrén, Florian Feldhaus, Boris Parák, and Zdenek Sustr. Open Cloud Computing Interface – JSON Rendering. Recommandation GFD-R-P.226, Open Grid Forum, October 2016.

[28] Francesco Palmieri, Massimo Ficco, Silvio Pardi, and Aniello Castiglione. A Cloud-based Architecture for Emergency Management and First Responders Localization in Smart City Environments. *Computers and Electrical Engineering*, 56(C):810–830, 2016.

[29] Jean Parpaillon, Philippe Merle, Olivier Barais, Marc Dutoo, and Fawaz Paraiso. OCCIware - A Formal and Tooled Framework for Managing Everything as a Service. In *Projects Showcase@ STAF'15*, volume 1400, pages 18–25, 2015.

[30] Louis Turnbull and Biswanath Samanta. Cloud Robotics: Formation Control of a Multi Robot System Utilizing Cloud Infrastructure. In *Proceedings of SoutheastCon*, pages 1–4, Jacksonville, USA, 2013.

[31] Lotfi Zaouche, Enrico Natalizio, and Abdelmadjid Bouabdallah. Ettaf: Efficient target tracking and filming with a flying ad hoc network. In *Proceedings of the 1st International Workshop on Experiences with the Design and Implementation of Smart Objects*, SmartObjects '15, pages 49–54, New York, NY, USA, 2015. ACM.

[14] https://www.iot-lab.info